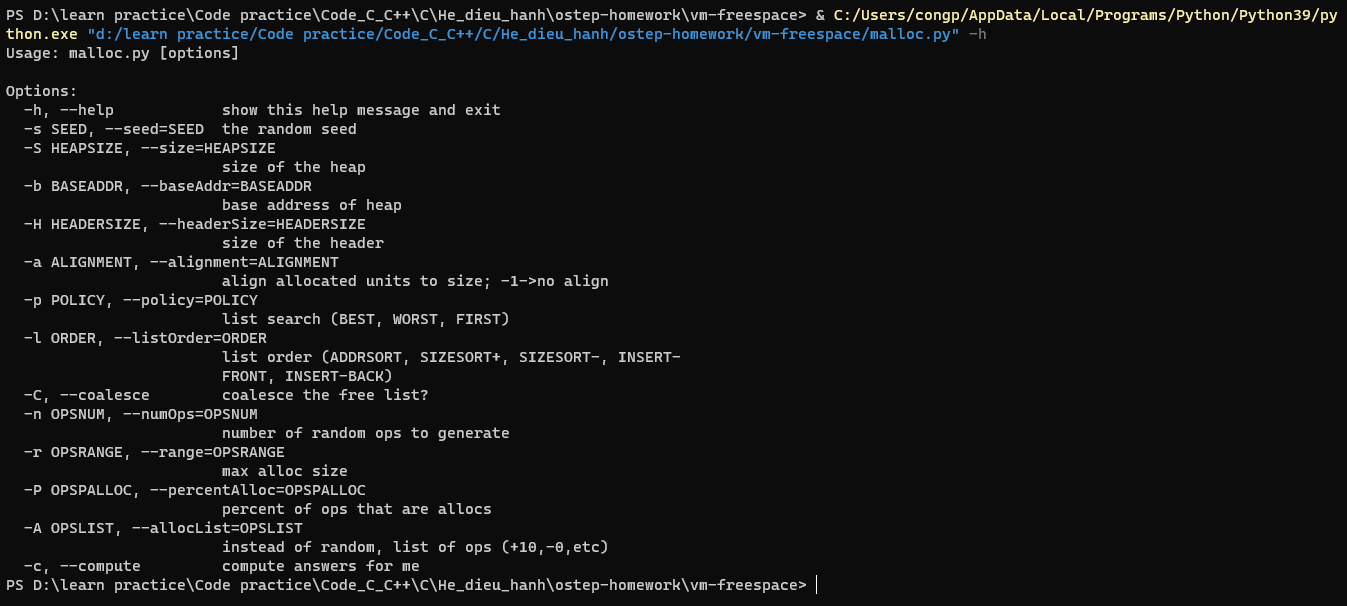
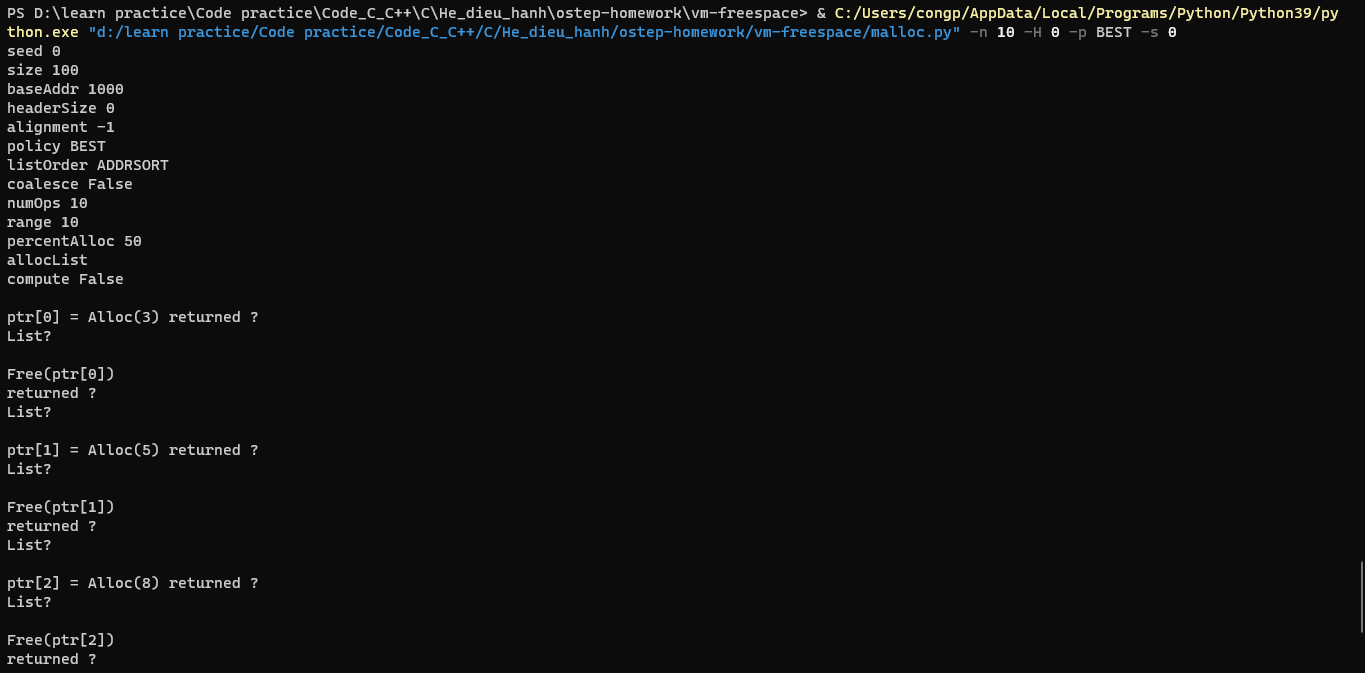
**Phần 1: free-space**

****

Câu 1: First run with the flags -n 10 -H 0 -p BEST -s 0 to generate a few random allocations and frees. Can you predict what alloc()/free() will return? Can you guess the state of the free list after each request? What do you notice about the free list over time?

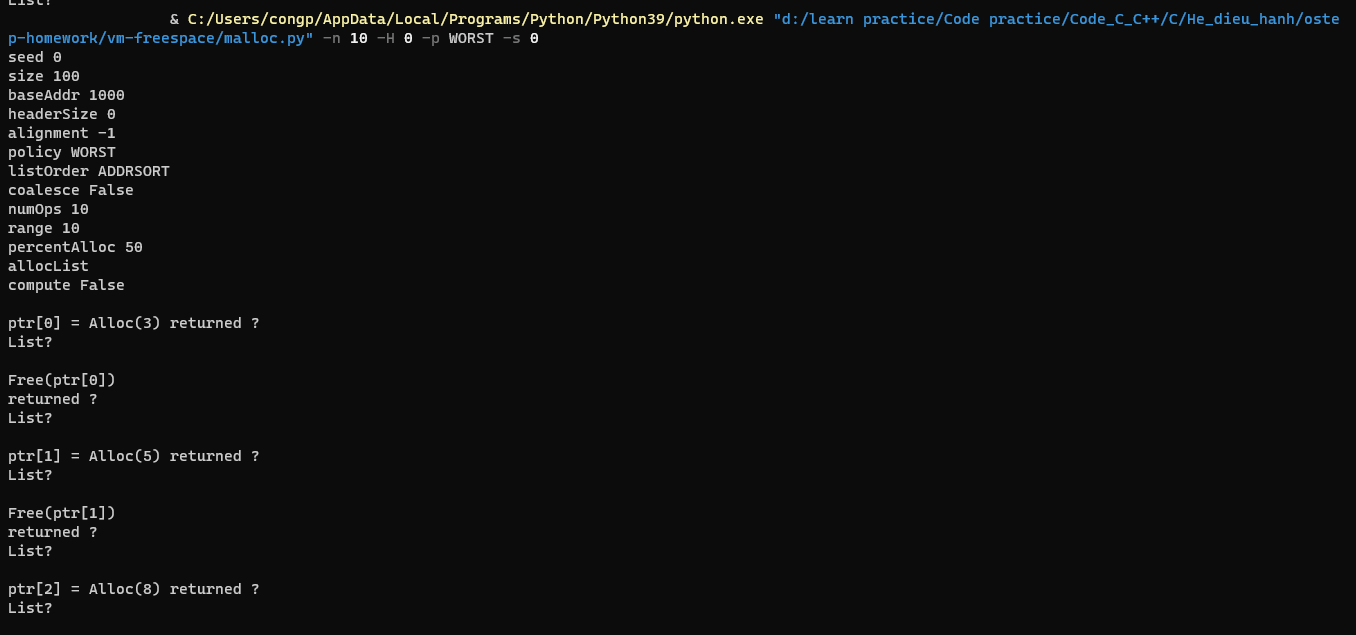


* Ta có thể dự đoán được nó nếu ta biết được những điều sau:

1. Base Address = 1000
2. Heap Size = 100

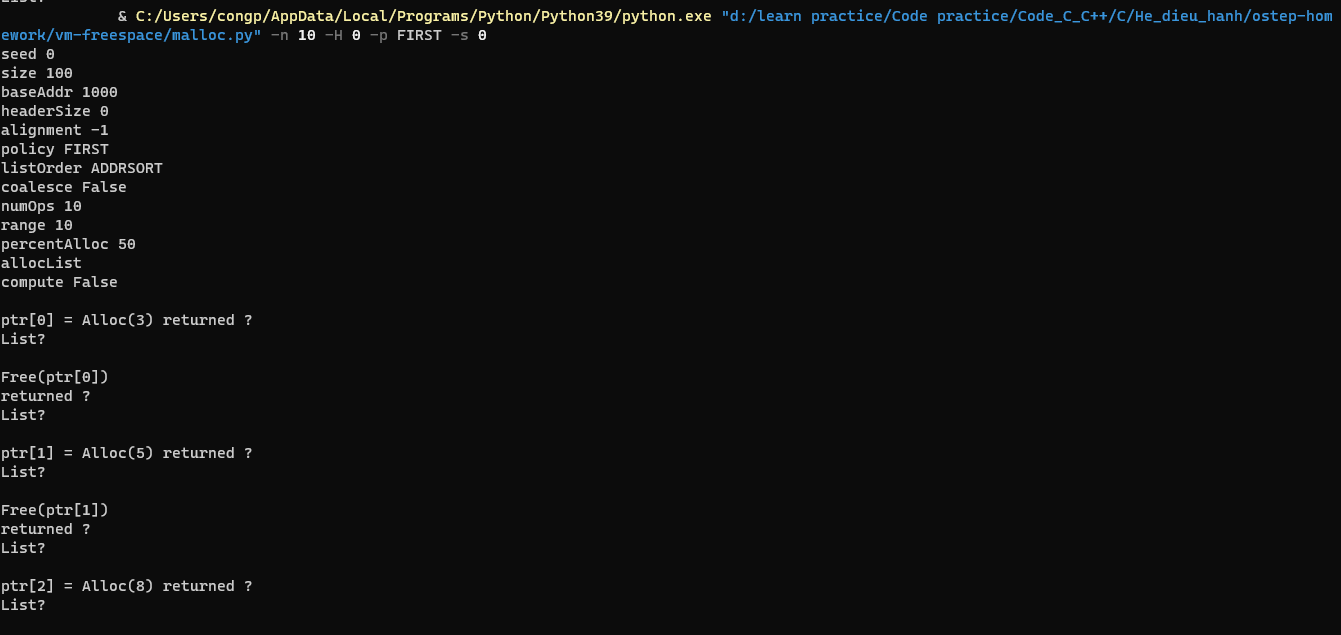
* Danh sách không gian trống sẽ bị nghẽn theo thời gian, với kích thước của khối khoonggian trống ngày càng nhỏ

Câu 2: How are the results different when using a WORST fit policy to search the free list (-p WORST)? What changes?



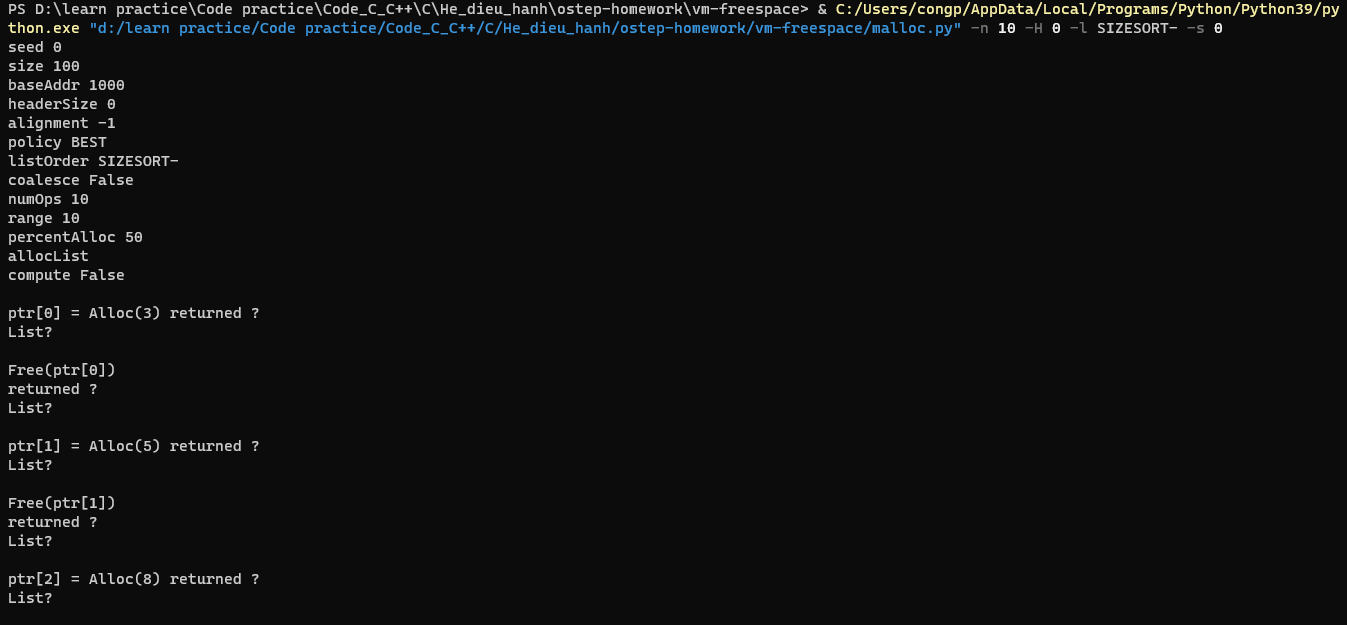
* Phân chia chính sách phù hợp nhất và sử dụng khối bộ nhớ lớn nhất hiện có.
* Kết quả là, các khối nhỏ được trả về đã không được sử dụng cho đến khi các khối bộ nhớ trống lớn hơn được chia nhỏ và sử dụng.
* ptr[0] = Alloc(3) returned 1000 List? [addr:1003, size:97]
* Free(ptr[0]) returned 0 List? [(addr:1000, size:3), (addr:1003, size:97)]
* ptr[1] = Alloc(5) returned 1003 List? [(addr:1000, size:3), (addr:1008, size:92)]
* Free(ptr[1]) returned 0 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:92)]
* ptr[2] = Alloc(8) returned 1008 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1016, size:84)]
* Free(ptr[2]) returned 0 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:8), (addr:1016,size:84)]
* ptr[3] = Alloc(8) returned 1016 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:8),(addr:1024, size:76)]
* Free(ptr[3]) returned 0 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:8), (addr:1016,size:8)(addr:1024, size:76)]
* ptr[4] = Alloc(2) returned 1024 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:8),(addr:1016, size:8)(addr:1026, size:74)]
* ptr[5] = Alloc(7) returned 1026 List? [(addr:1000, size:3), (addr:1003, size:5), (addr:1008, size:8),(addr:1016, size:8)(addr:1033, size:67)]

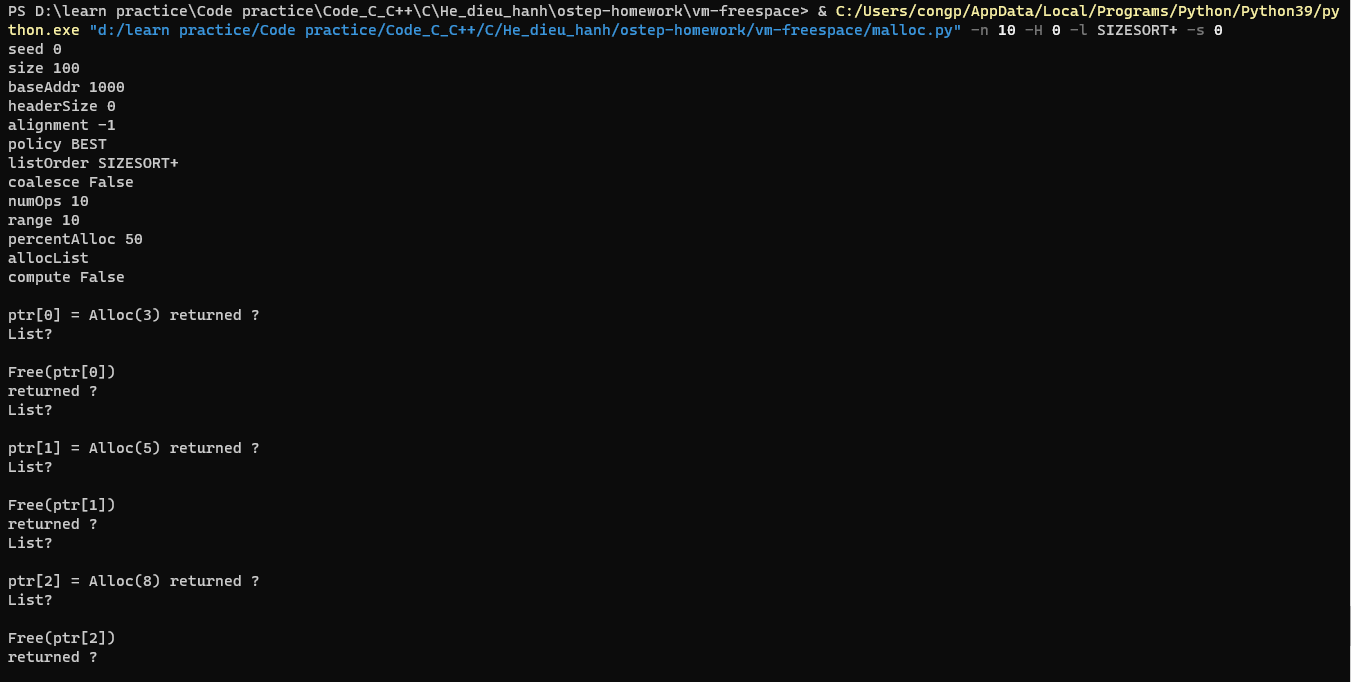
Câu 3: What about when using FIRST fit (-p FIRST)? What speeds up when you use first fit?

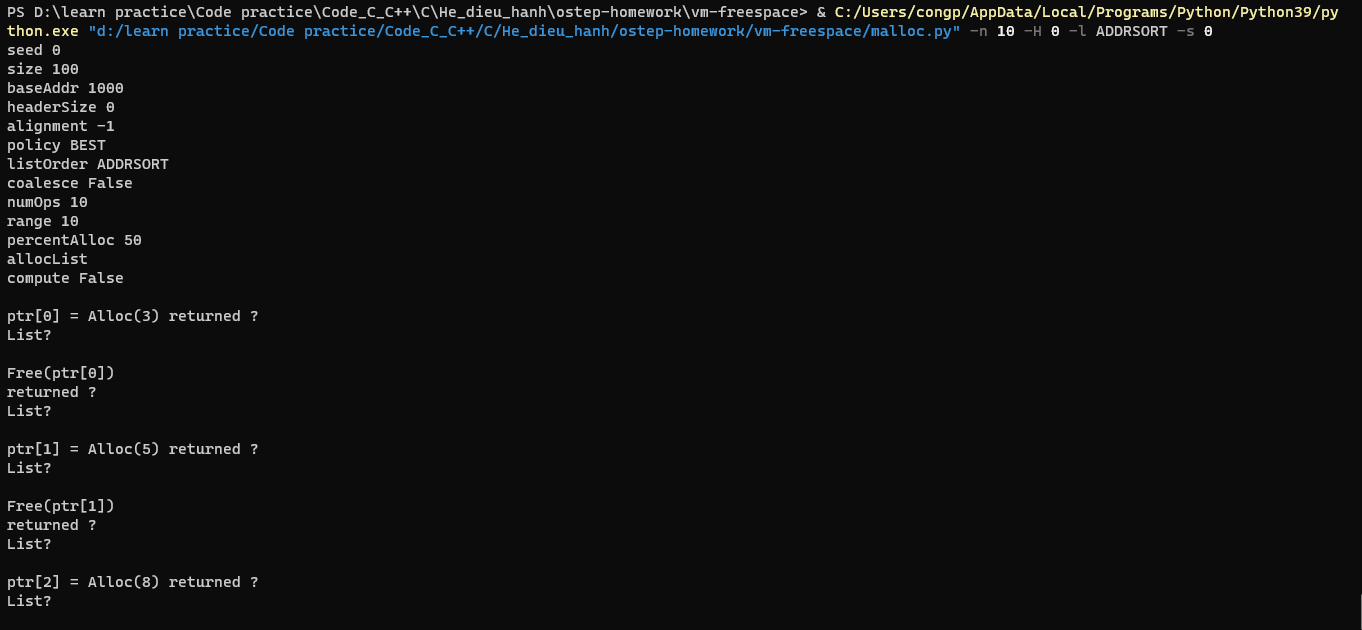


* Việc tra cứu để tìm không gian trống sẽ tăng tốc khi chúng ta sử dụng phù hợp đầu tiên. Bởi vì, chúng ta không phải kiểm tra tất cả các vị trí có sẵn. Chúng ta chỉ cấp phát khối bộ nhớ phù hợp đầu tiên cho bộ nhớ yêu cầu công việc trong heap.

Câu 4:

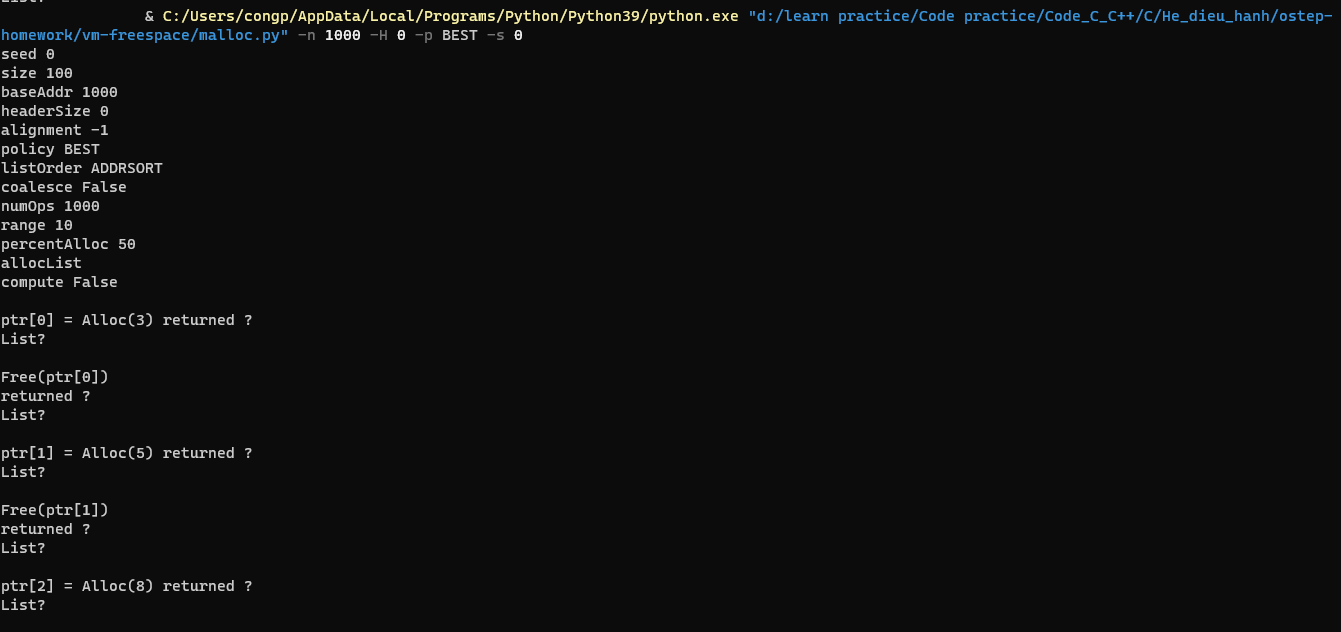






* Best: Trong SIZESORT +, chúng tôi không cần phải xem qua toàn bộ danh sách và chúng tôi có thể dừng lại khi chúng tôi tìm thấy phần tử lớn nhất đầu tiên. Vì vậy, SIZESORT + tăng tốc độ TỐT NHẤT.
* Work: Trong SIZESORT-, đầu tiên chúng ta lấy phần lớn nhất của bộ nhớ, do đó thời gian cấp phát bộ nhớ là O (1).
* First: SIZESORT- là nhanh nhất, vì mỗi cuộc gọi đầu tiên sẽ cấp một phân bổ bộ nhớ có kích thước thích hợp hoặc không thành công.

Câu 5: Coalescing of a free list can be quite important. Increase the number of random allocations (say to -n 1000). What happens to larger allocation requests over time? Run with and without coalescing (i.e., without and with the -C flag). What differences in outcome do you see? How big is the free list over time in each case? Does the ordering of the list matter in this case?

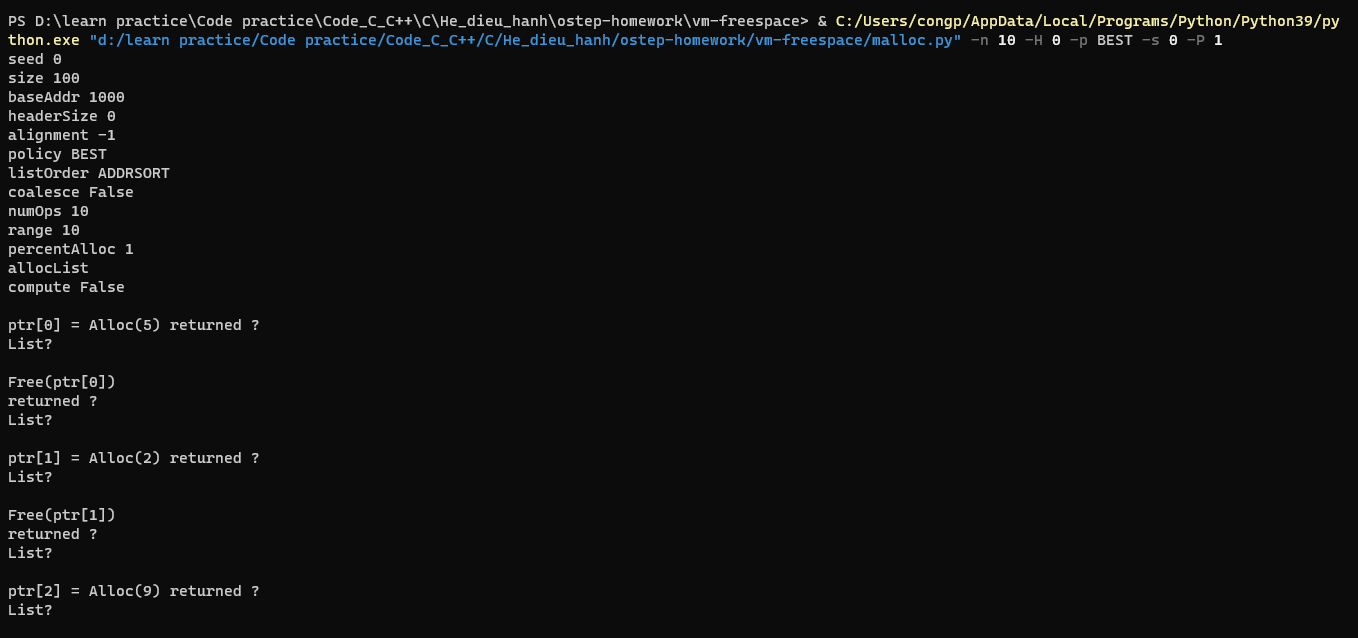


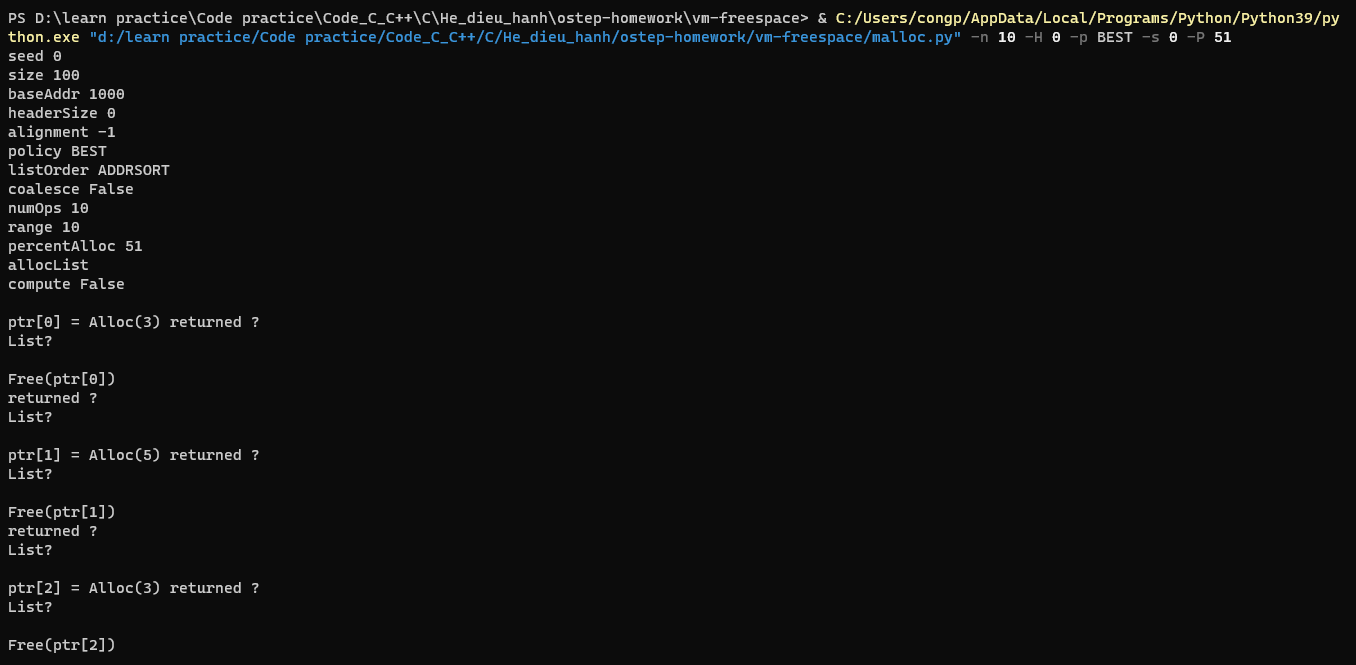
…

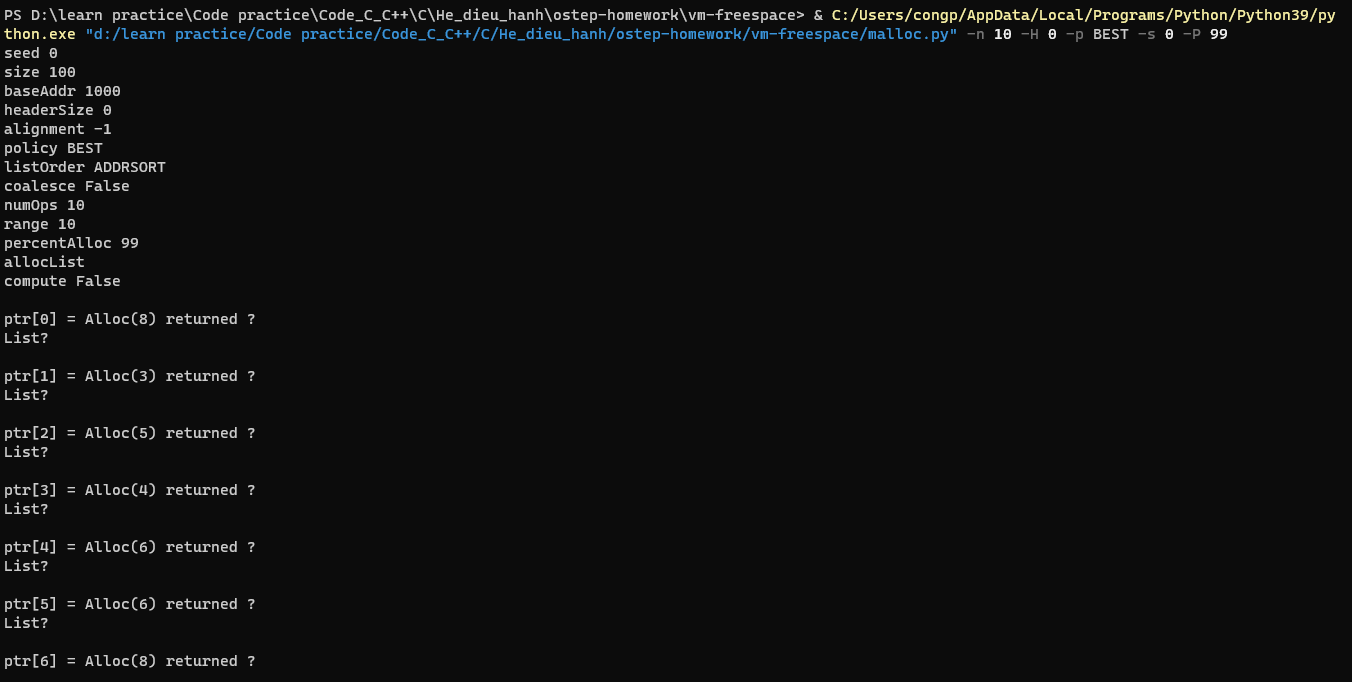


* Yêu cầu phân bổ lớn hơn không thành công mà không liên kết với nhau trong mem miễn phí. bởi vì danh sách miễn phí sẽ được tải với các mem miễn phí nhỏ. các khối. Danh sách miễn phí trở nên bị tắc nghẽn với một loạt các khối bộ nhớ trống ít ỏi do thiếu liên kết. Trong thực tế tất cả các chiến lược tra cứu bộ nhớ trống, cách gợi ý này cũng làm tăng thời gian tra cứu. Chỉ trong lần đầu tiên phù hợp miễn phí mem. chính sách vị trí có quan trọng thứ tự của danh sách không.

Câu 6: What happens when you change the percent allocated fraction -P to higher than 50? What happens to allocations as it nears 100? What about as the percent nears 0?

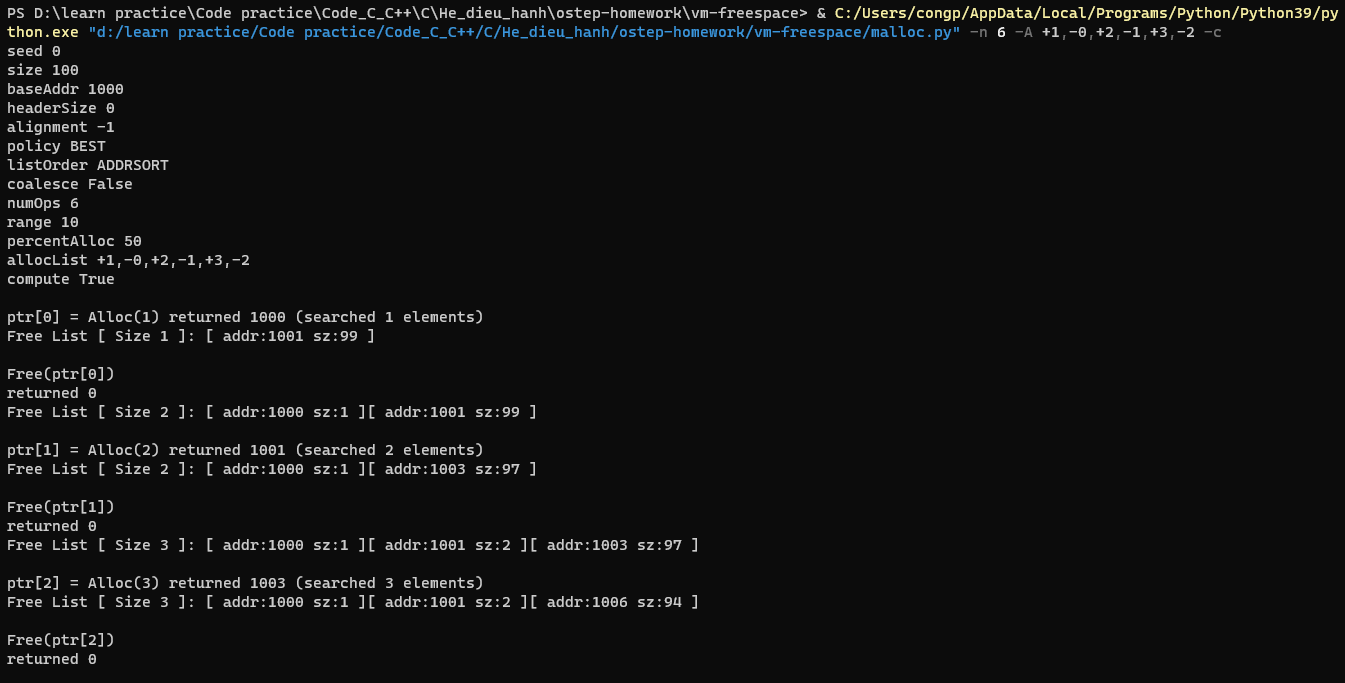






* Hoạt động phân bổ sẽ là hoạt động đống phổ biến nhất. Khi chúng tôi tăng phần trăm được phân bổ lên khoảng 100 hoặc 100, chúng tôi sẽ có được kết quả tốt hơn. Hầu hết mọi hoạt động sau đó sẽ là một hoạt động phân bổ. Khi chúng ta tiến gần hơn đến con số 0, việc phân bổ và các hoạt động miễn phí sẽ hoàn thành trong một nửa thời gian. Chúng tôi không thể giải phóng RAM chưa được cấp phát. Để giải phóng bộ nhớ, trước tiên chúng ta phải cấp phát một số.

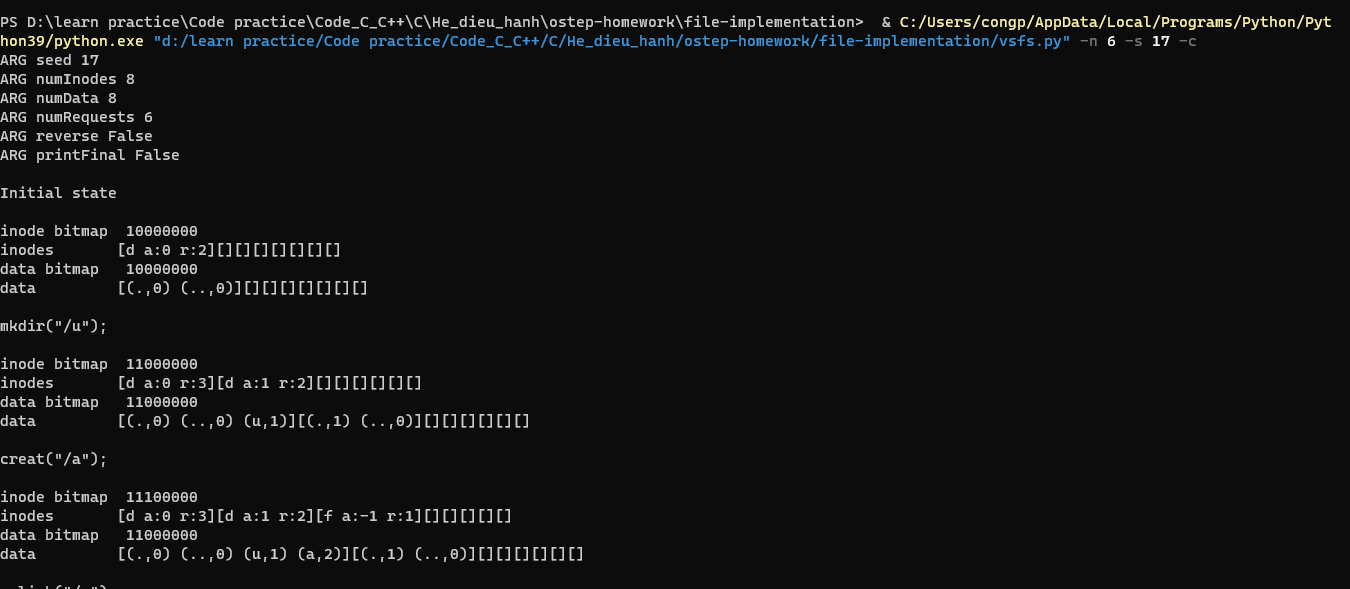
Câu 7: What kind of specific requests can you make to generate a highlyfragmented free space? Use the -A flag to create fragmented free lists, and see how different policies and options change the organization of the free list.

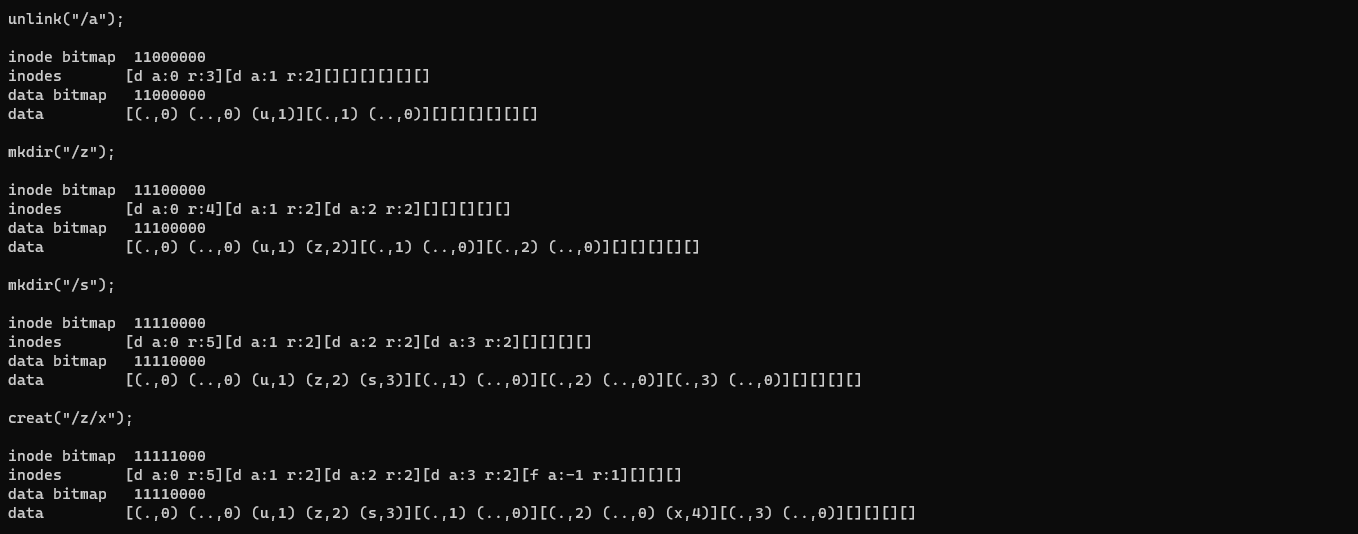


**Phần 2: File System Implementation**

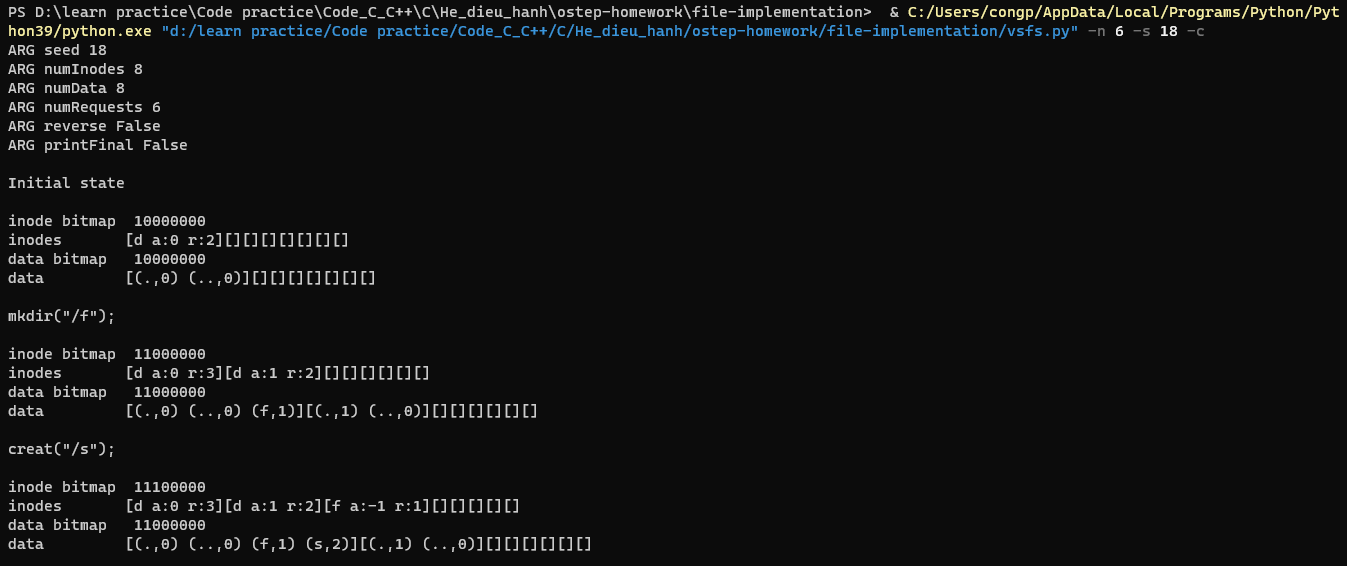
Câu 1: Run the simulator with some different random seeds (say 17, 18, 19, 20), and see if you can figure out which operations must have taken place between each state change

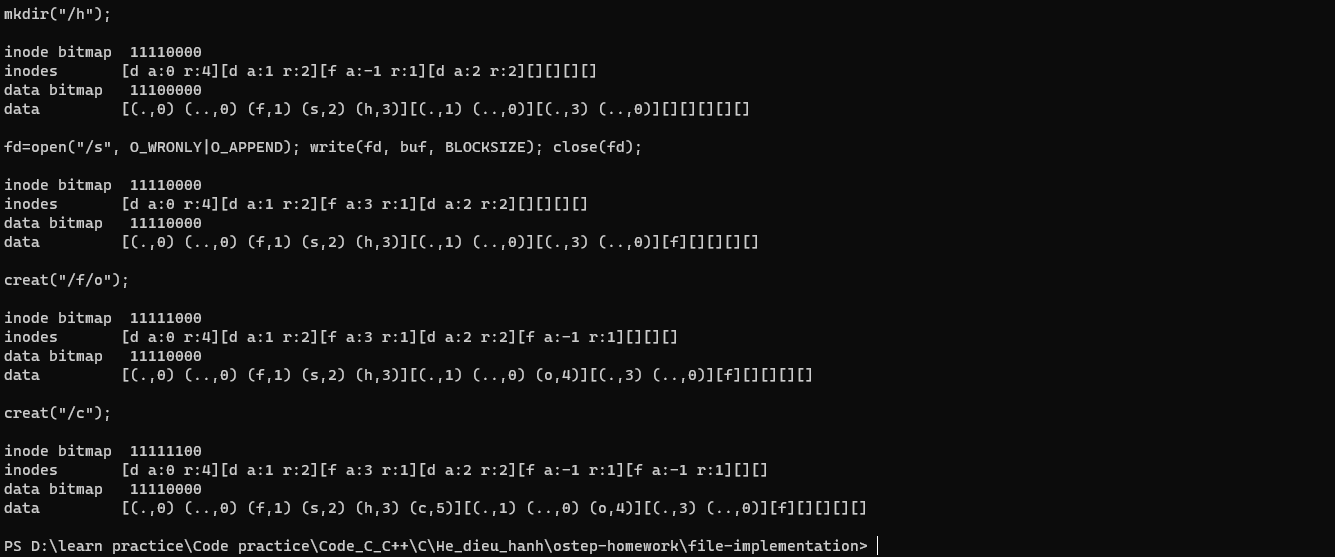
-17:



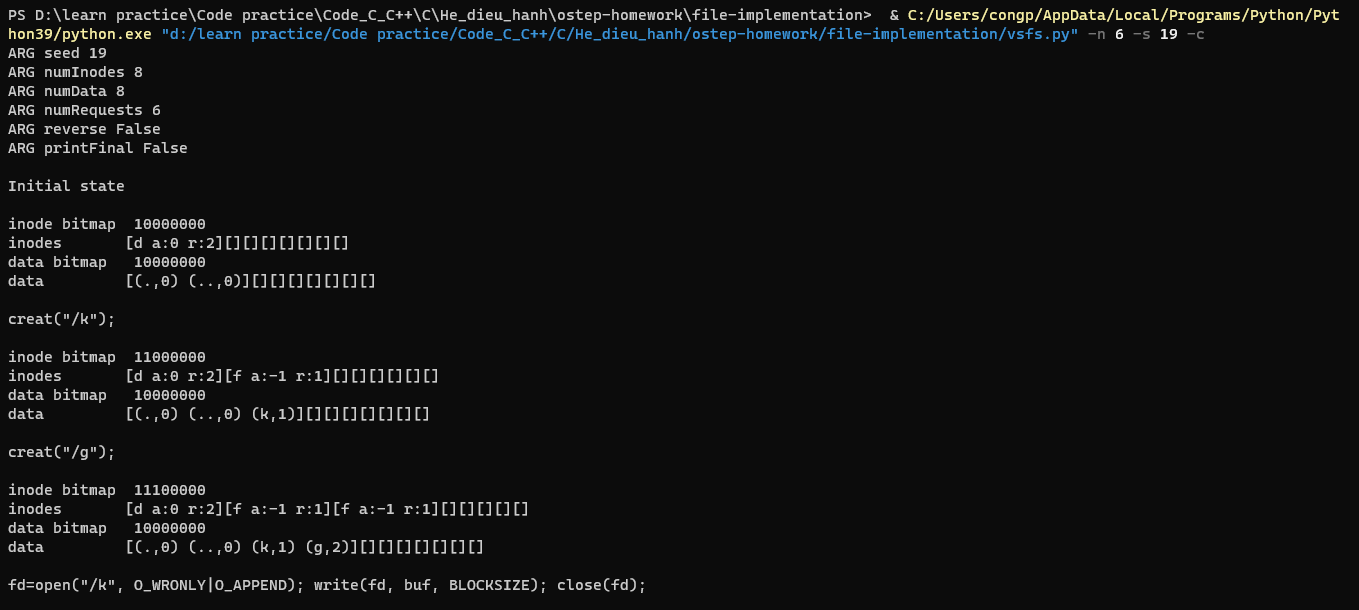
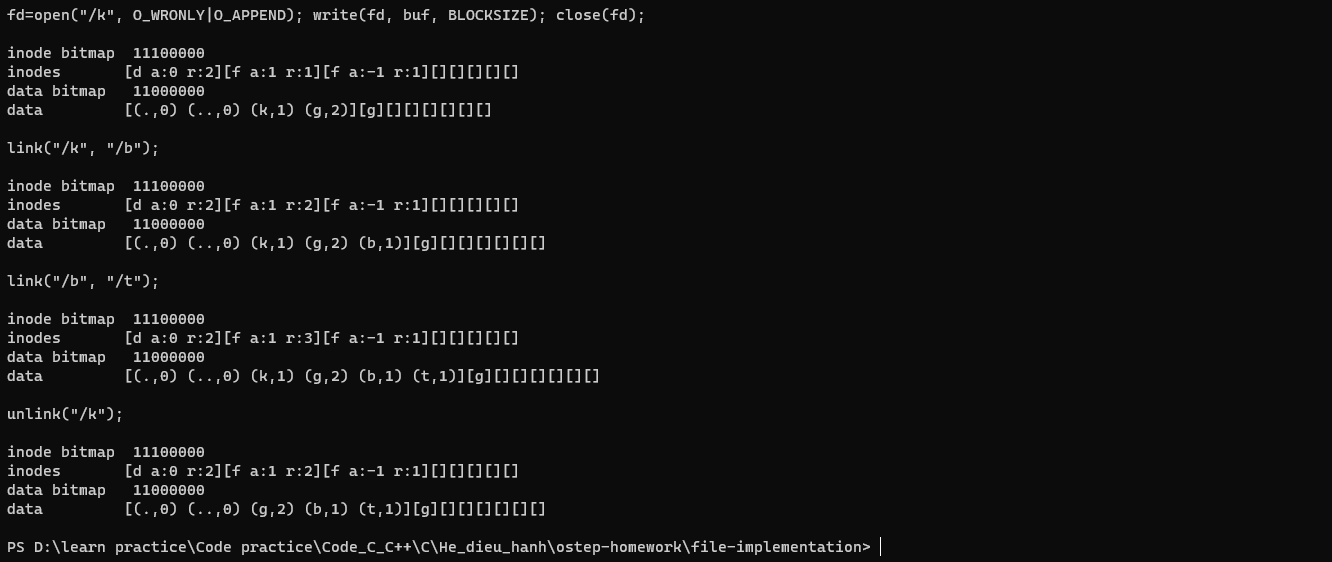


-18:





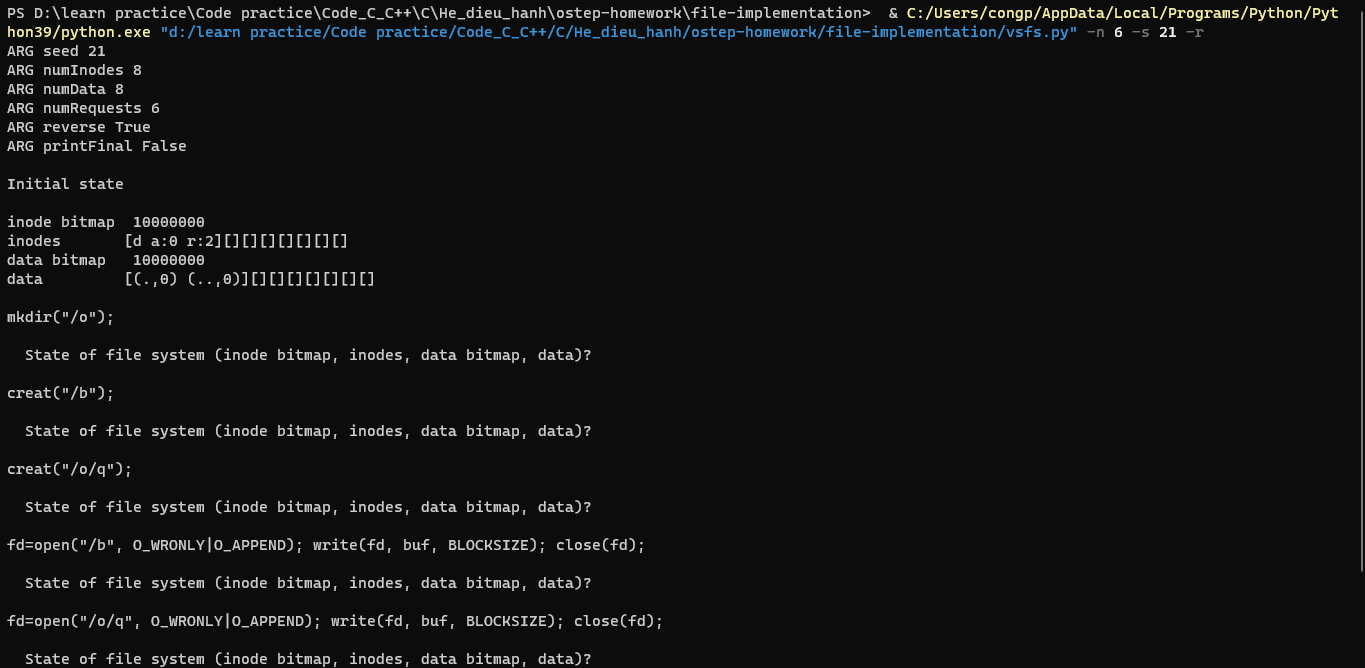
-19:

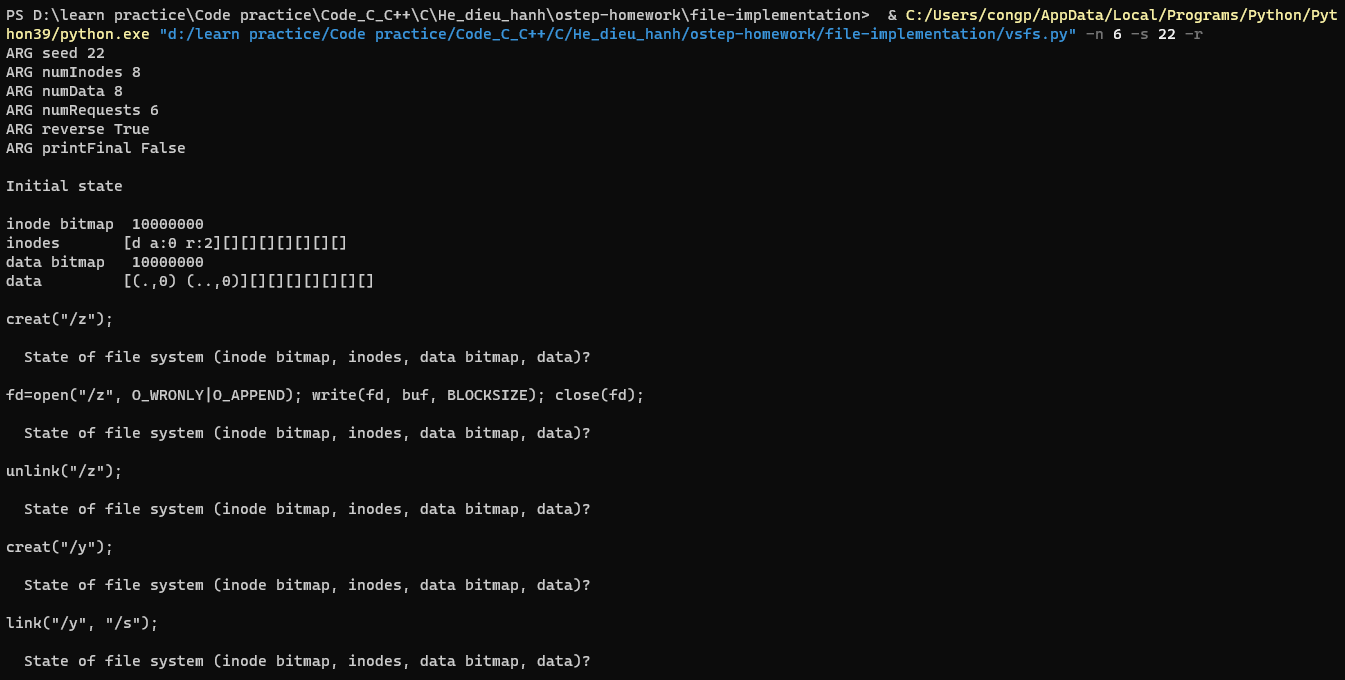
* Các file được tạo ra khác nhau.
* Các inodes sẽ khác nhau vì nó phụ thuộc vào các thư mục mà chúng tao ra và ở trong nó
* Các data bitmap ở 18 với 19 khác nhau do nó không chung 1 file
* ở phần 19 sẽ có link và unlink

Câu 2: Now do the same, using different random seeds (say 21, 22, 23, 24), except run with the -r flag, thus making you guess the state change while being shown the operation. What can you conclude about the inode and data-block allocation algorithms, in terms of which blocks they prefer to allocate?

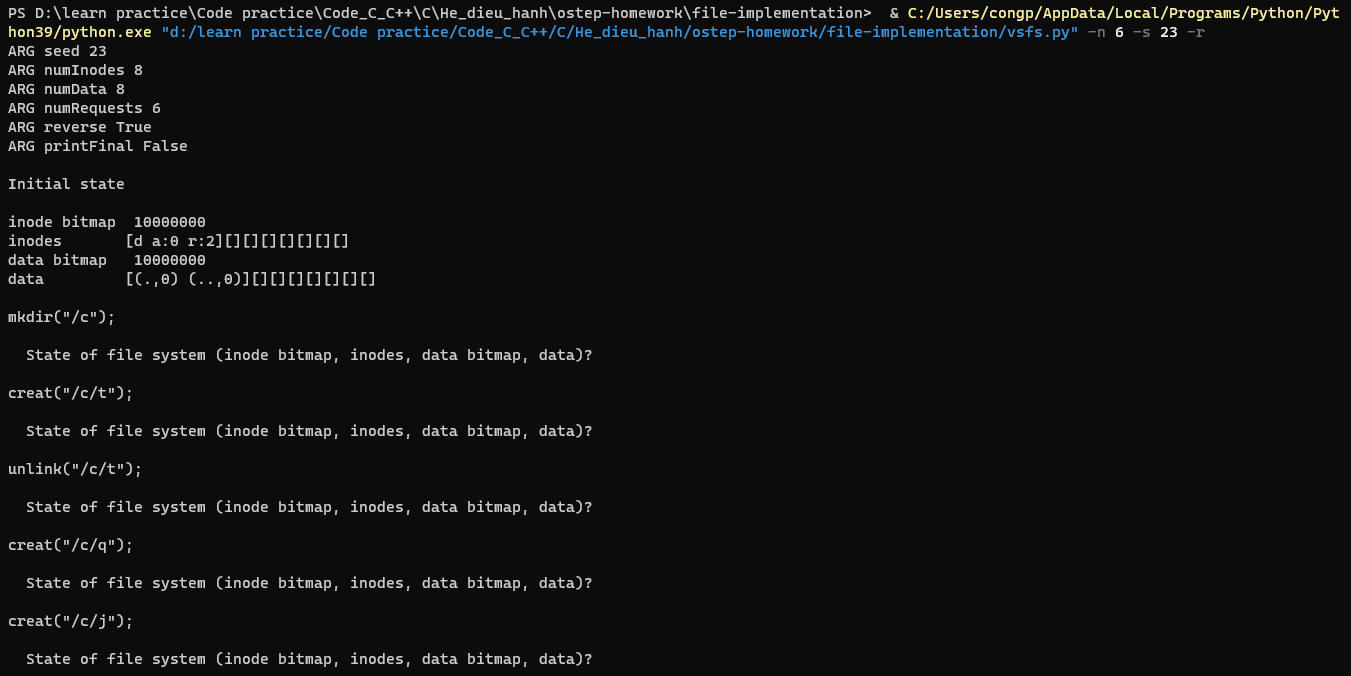
* 21



* 22:

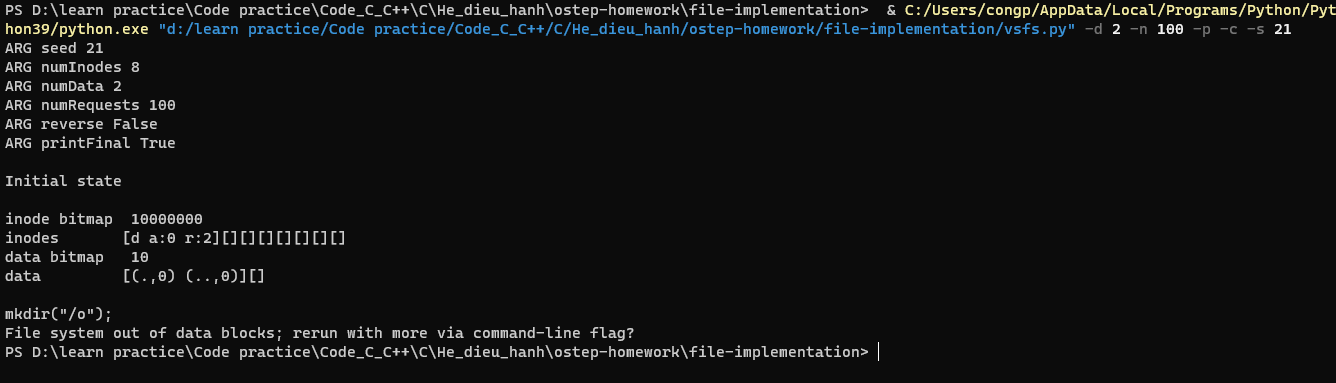


* 23:

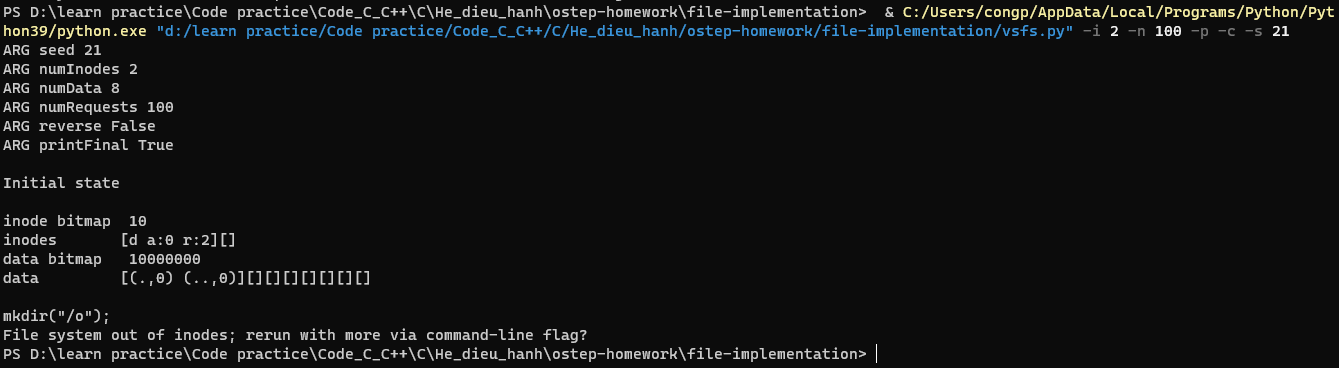


Các seed càng lớn sẽ có đường dẫn càng phức tạp

Câu 3: Now reduce the number of data blocks in the file system, to very low numbers (say two), and run the simulator for a hundred or so requests. What types of files end up in the file system in this highlyconstrained layout? What types of operations would fail?

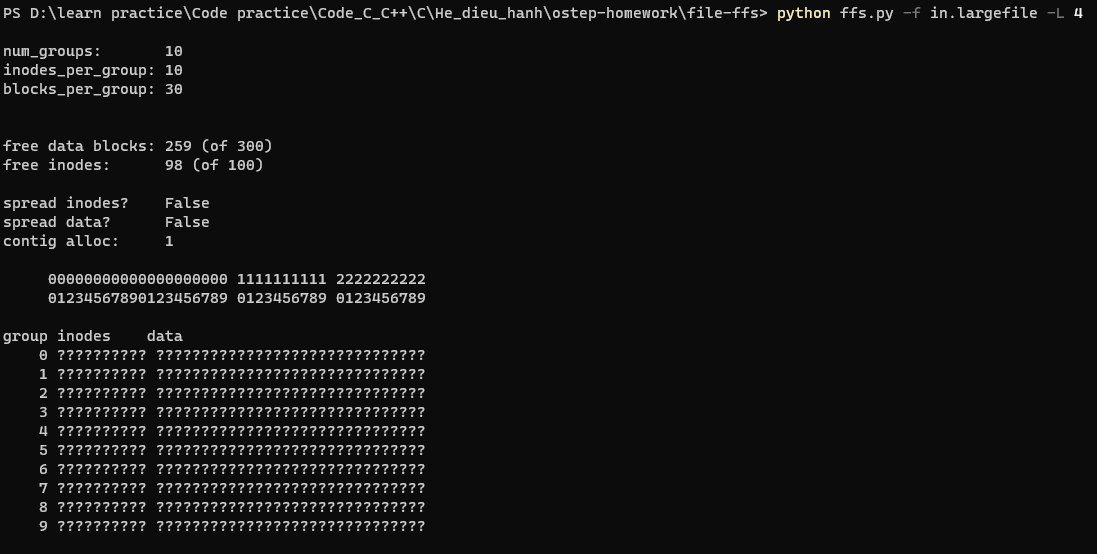


Câu 4: Now do the same, but with inodes. With very few inodes, what types of operations can succeed? Which will usually fail? What is the final state of the file system likely to be?

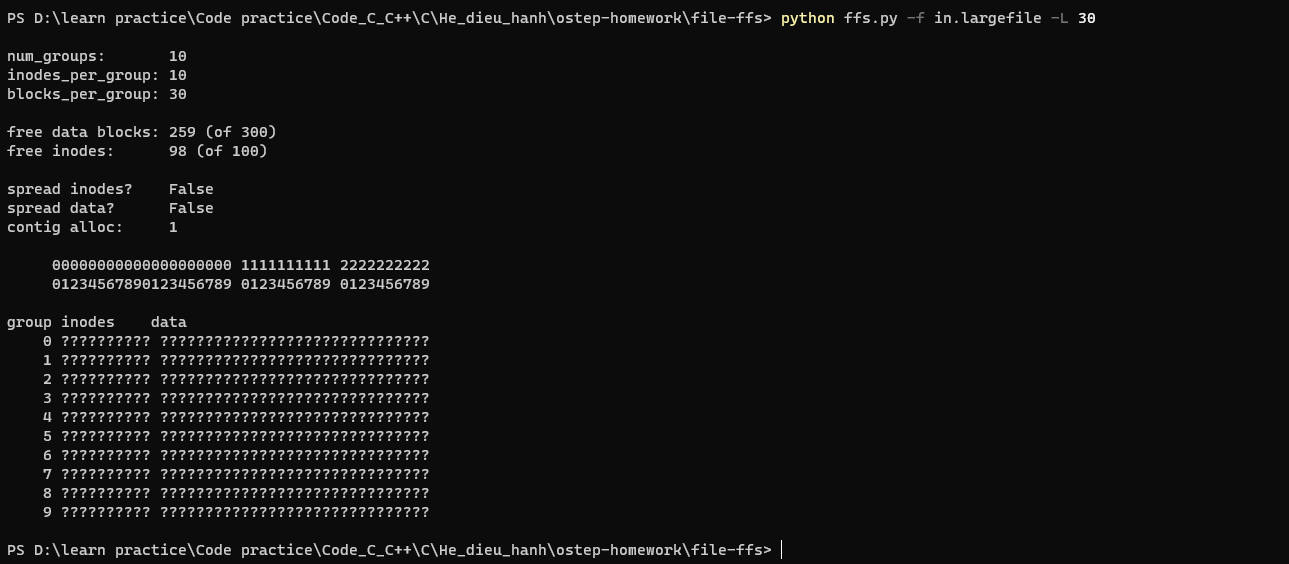


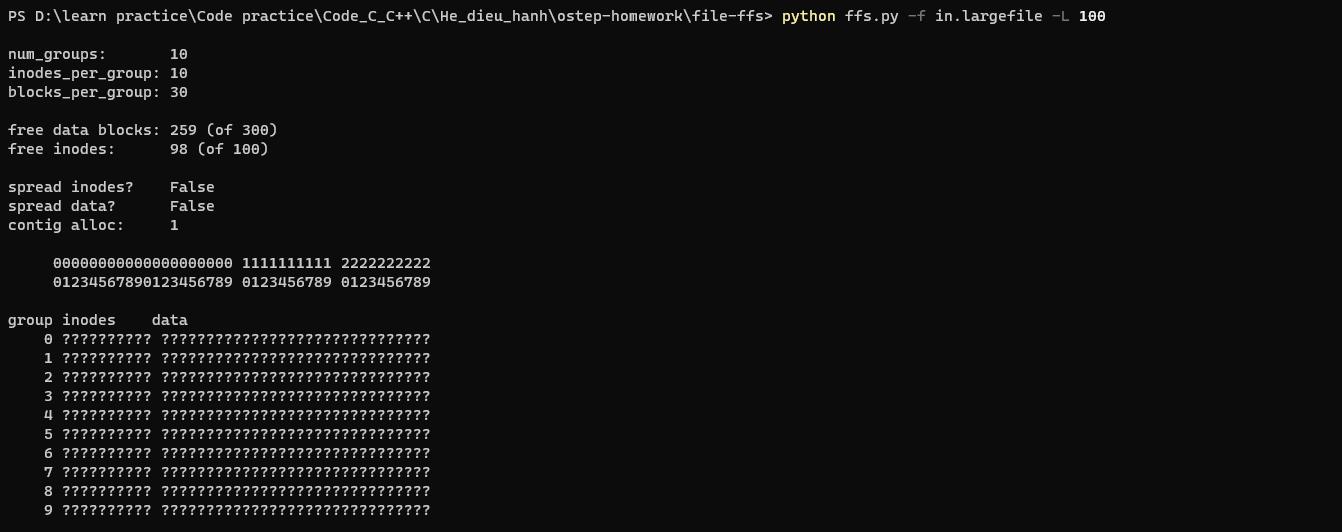
**Phần 3: Locality and The Fast File System**

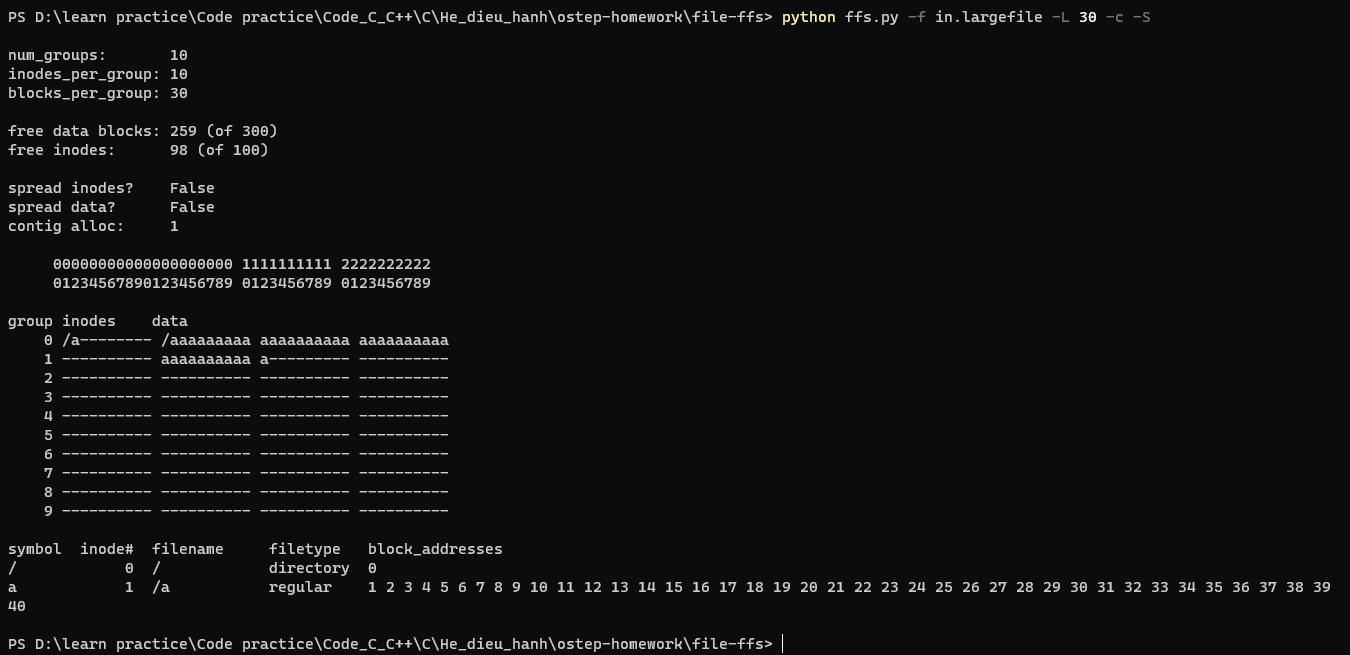
Câu 1:



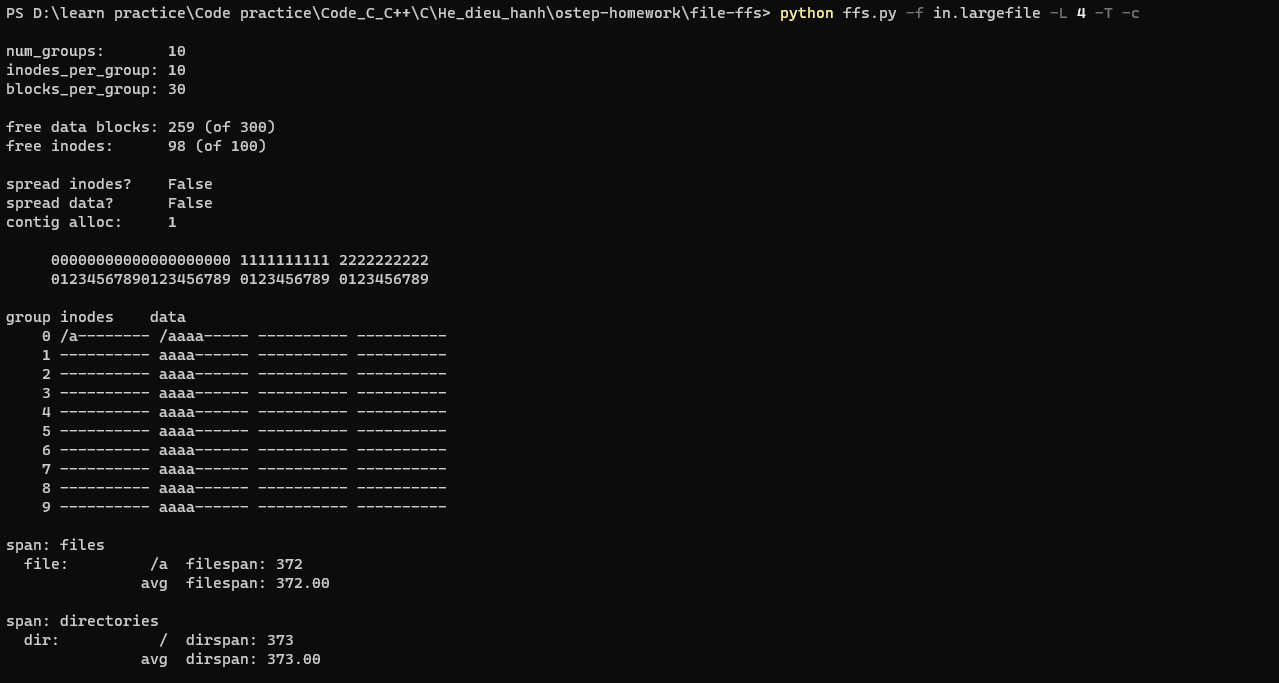
Câu 2:



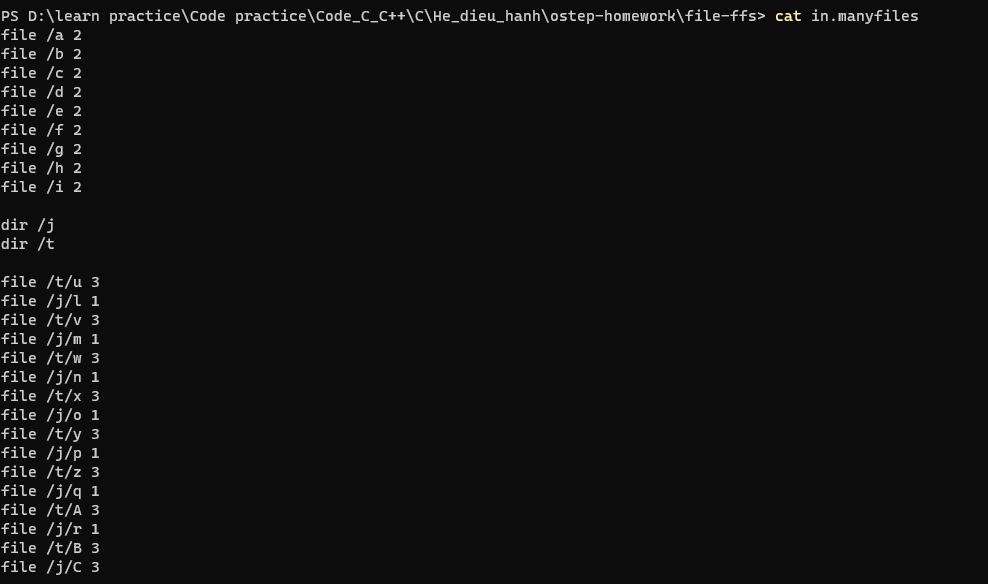




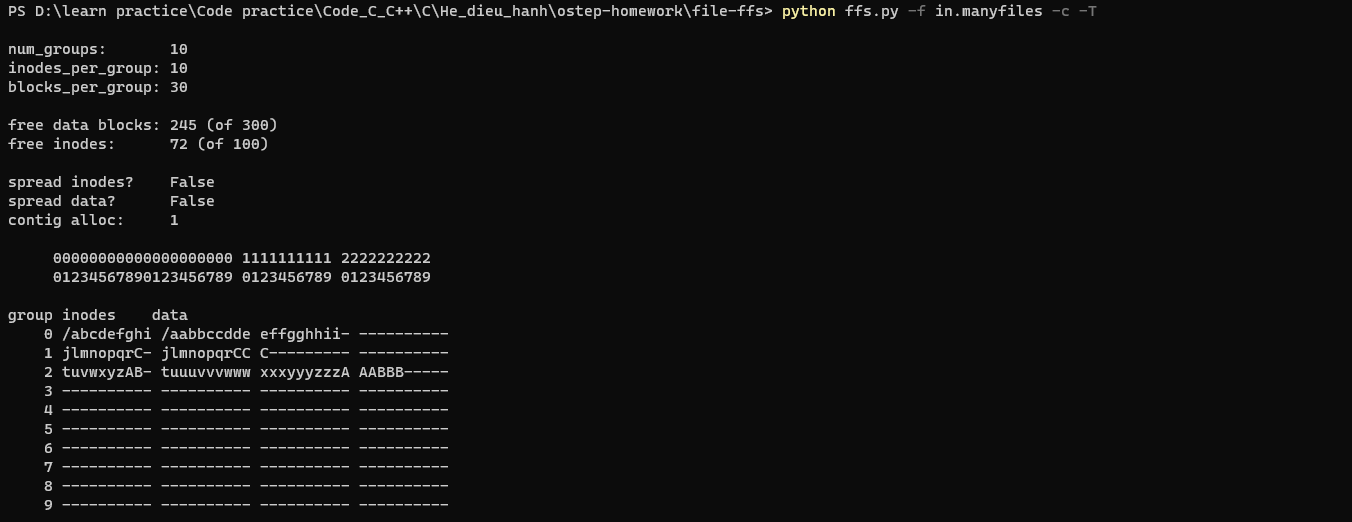
Câu 3:



Câu 4:

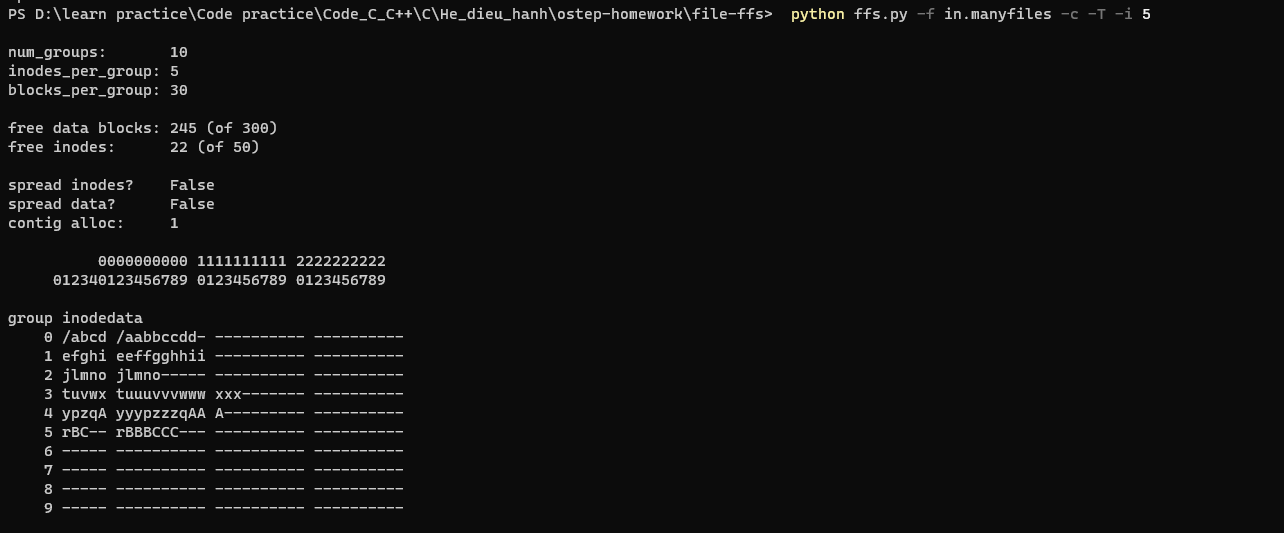


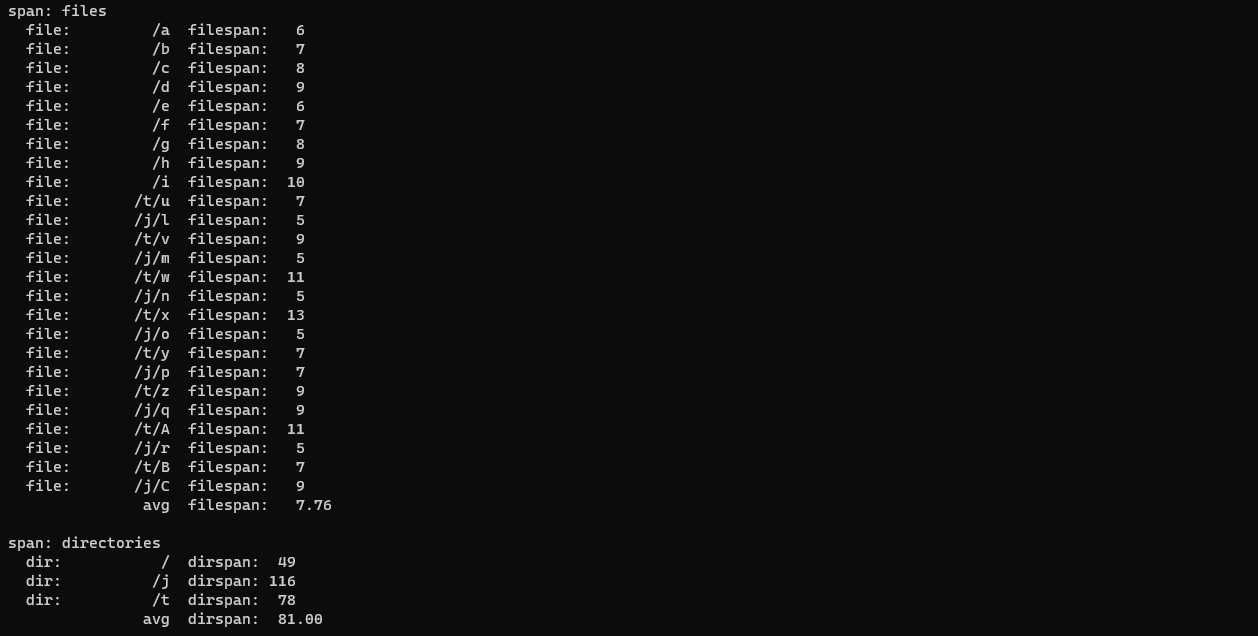
Câu 5:



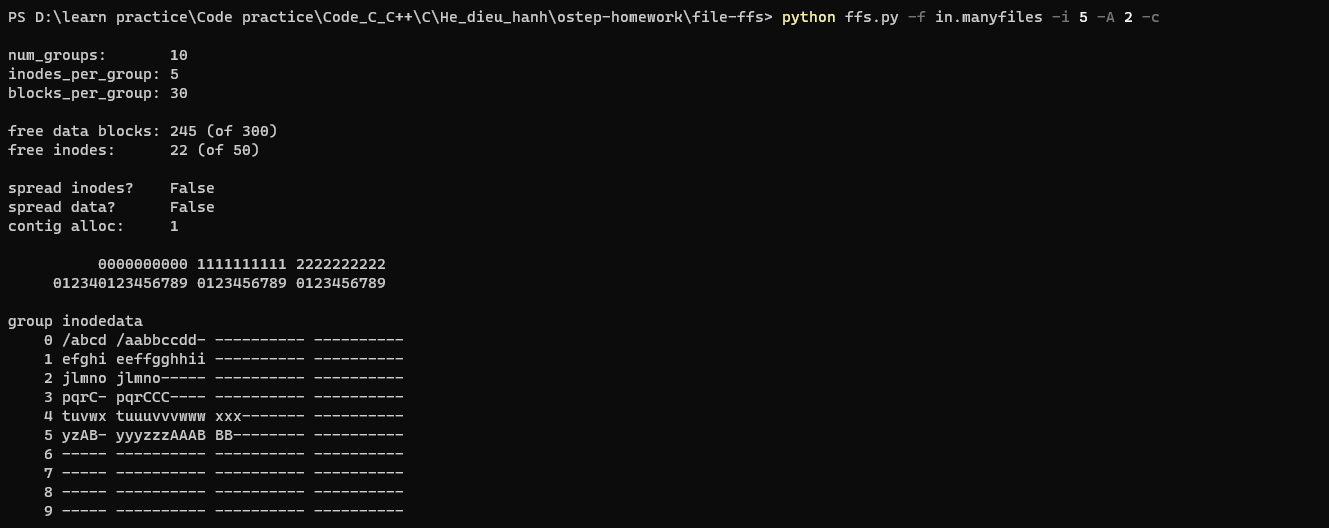


Câu 6:

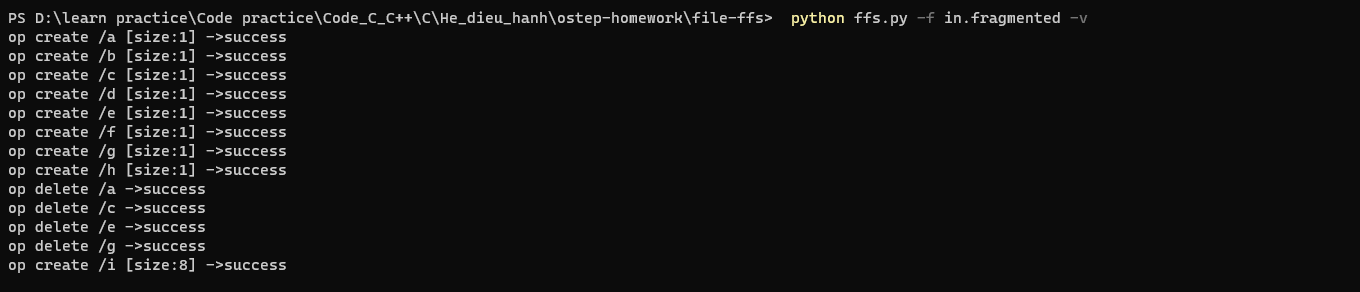


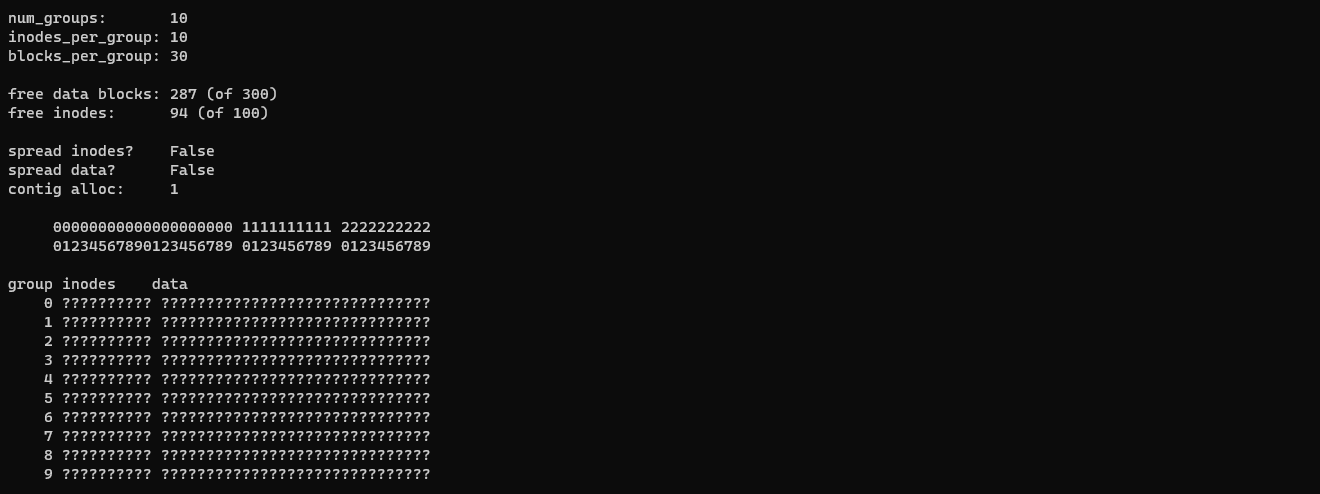


Câu 7:



Câu 8:





Câu 9:

